In car embedded electronic architectures: how to ensure their safety

Françoise Simonot-Lion

IN CAR EMBEDDED ELECTRONIC ARCHITECTURES: HOW TO ENSURE THEIR SAFETY

Françoise Simonot-Lion

LORIA – INPL (UMR CNRS 7503)
2, avenue de la Forêt de Haye – 54516 Vandoeuvre- lès-Nancy Cedex France
Tel. +33 3 83 59 55 79
Fax. +33 3 83 59 56 62
simonot@loria.fr

Abstract: The part of software based systems in a car is growing. Moreover, in the next years will emerge the X-by-Wire technology that intends to replace mechanical or hydraulic systems by electronic ones even for critical function as braking or steering. This requires a stringent proof that these new vehicles will ensure the safety of driver, occupants, vehicle and environment. In this paper, we intend to list certain activities and key points for ensuring the development of a safe and optimized embedded system. More precisely, we propose two main axis that contribute to establish a design methodology of such systems. The first one identifies the generic components of an embedded system while the second one details how to model and validate the embedded system throughout the different steps of the development process.

Keywords: embedded systems, safety, real time, validation, architecture description language

1. INTRODUCTION

While automobile production is likely to increase slowly in the coming years (42 millions cars produced in 1999 and only 60 millions planned in 2010), the part of embedded electronics and more precisely embedded software is growing. The cost of electronic systems was $37bn in 1995 and $60bn in 2000, with an annual growth rate of 10%.

The reasons for this evolution are technological as well as economical. The cost of hardware components is decreasing while their performances and reliability are increasing. The emergence of embedded fieldbuses leads to a significant reduction of the wiring cost. Finally, software technology facilitates the introduction of new functions whose development would be costly or even not feasible if using only mechanical or hydraulic technology and allows therefore satisfying the end user requirements in terms of safety and comfort.

Who is concerned by this evolution? First the *vehicle customer*, for which the requirements are on the one hand, the increase of performance, comfort, assistance for mobility efficiency (navigation), … and on the other hand, the reduction of vehicle consumption and cost. Furthermore he requires a reliable embedded electronic system that ensures safety properties. Secondly, the stakeholders, *car makers* and *suppliers*, who are interested in the reduction of time to market, development cost, production and maintenance cost. Finally this evolution has a strong *impact on the society*: legal restrictions on exhaust emission, protection of the natural resources and of the environment, …

In this presentation, we intend to list certain activities and key points for ensuring the development of a safe and "optimal" embedded system that is suited to the above mentioned requirements. For this purpose, we develop the context and the problematic in section 2. Then in sections 3 and 4, we propose two main axis

that contribute to establish a design methodology of such systems. The first one identifies the generic components that compose an embedded system while the second one details how the embedded system is modelled and validated along the different steps of the development process.

2. CONTEXT

2.1. Several domains and specific problems

Traditionally, an in-car embedded system is divided in four domains that correspond to different functionalities, constraints and models. Two of them are concerned specifically with safety : “power train” and “chassis” domain. The third one, “body”, is emerging and presently integrated in major of cars. And finally, “telematic” and “Human Machine Interface” domains take benefit of continuous progress in the field of multimedia and internet.

Power train

This domain represents the system that controls the motor according to explicit solicitations of the driver (speeding up, slowing down, …), implicit solicitations of the driver (driving facilities, fuel consumption, …) and environmental constraints (exhaust pollution, noise, …). Moreover, this control has to take into account requirements from other parts of the embedded system as climate control or ESP (Electronic Stability Program). Traditional tools dedicated for general control command applications are used for the development of power train systems (Matlab / Simulink, for example and simulation approach).

In this domain, the main characteristics are:

- at a functional point of view: different control laws, complex control laws (multi-variables), different sampling periods, …
- at a hardware point of view: specific sensors (minimisation of the criteria “cost / resolution”), high computation power, high storage capacities, dedicated coprocessors (floating point computations),
- at an implementation point of view: several tasks with different activation rules (different periods), stringent time constraints imposed to task scheduling, mastering safe communications with other systems and with local sensors / actuators.

Chassis

It gathers systems as ABS (Antilock Braking System), ESP (Electronic Stability Program), ASC (Automatic Stability Control), 4WD (4 Wheel Drive), … that control the chassis components (wheel, suspension, …) according to requirements as steering, braking solicitations and several forces (ground, wind, …).

The characteristics of the chassis domain and the underlying models are similar to those presented for power train domain.

Body

Wipers, lights, doors, windows, seats, mirrors are more and more controlled by software based systems. This kind of functions make up the body domain. In this case, there are numerous functions, some of them being critical. They imply globally many communications between them and consequently a complex distributed architecture. There is an emergence of the notion of sub-system or sub-cluster based on low cost fieldbuses as, for example, LIN that connect modules realized as integrated mechatronic systems. On another side, the body domain integrates a central subsystem, termed the “central body electronic” whose main functionality is to ensure message transfers between different systems or domains. This system is recognized to be a central critical entity.

Body domain implies mainly discrete event applications. Their design and validation rely on state transition models (as SDL, Statecharts, UML state transition diagrams). These models allow, mainly by simulation, the validation of a functional specification. Their implementation implies a distribution over a complex hierarchical hardware architecture. High computation power for the central body electronic entity, fault tolerance and reliability properties are imposed to the body domain systems. A challenge in this context is first to be able to develop exhaustive analyze of state transition diagrams and second, to ensure that the implementation respects the fault tolerance and safety constraints. The problem here is to achieve a good balance between time triggered approach and flexibility.

Telematic and HMI (Human Machine Interface)

Next generation of telematic devices provides new sophisticated Human Machine Interfaces to the driver and the other occupants of a vehicle. They enable not only to communicate with other systems inside the vehicle but also to exchange information with the external world. Such devices will be in the future upgradeable and for this domain, a “plug and play” approach has to be favoured.

These applications have to be portable and the services furnished by the platform (operating system

and / or middleware) has to offer generic interfaces and downloading facilities. The main challenge here is to preserve the security of the information from, to or inside the vehicle. Sizing and validation do not relies on the same methods than for the other domains. Here we shift from considering messages, tasks and deadline constraints to fluid data streams, bandwidth sharing and multimedia quality of service.

2.2. A cooperative development process

Strong co-operation between suppliers and carmakers in the design process implies the development of a specific concurrent engineering approach. In order to specify this process, synchronisation points (rendezvous) across the co-operative development model have to be identified and the information exchanged at these points must be characterised. Furthermore, an unique syntax of the exchanged information has to be defined and the development of reusable components is a main way for cost reduction.

2.3. The emergence of X-by-Wire technology

At present some critical functions are realised by software-based systems, as braking assistance, active suspension, steering functionalities etc.. They are subject to stringent timing constraints and more generally to dependability constraints. In the close future, these constraints will be more critical with the generalisation of X-by-Wire technology. X-by-Wire is a generic term used when mechanical and / or hydraulic systems are replaced by "electronic" ones (intelligent devices, networks, computers supporting software components that implement filtering, control, diagnosis, … functionalities). For example, we can cite brake-by-wire, steer-by-wire, that will be shortly integrated in cars for the implementation of critical and safety relevant functions.

Therefore the development of such systems must define an feasible system, i.e. satisfying these constraints. Conventional mechanical and hydraulic systems have stood the test of time and have proved to be reliable; it is not the same for critical software based systems. In aerospace / avionic industries, X-by-Wire technology is currently employed; but, for ensuring safety properties, are used specific hardware and software components, specific fault tolerant solutions (heavy and costly redundancies of networks, sensors and computers) and certified design and validation methods. Now there is a challenge to adapt these solutions to automotive industries that impose stringent constraints on component cost, electronic architecture cost (minimisation of redundancies) and development length. Consequently, there is a real need for mastering the cooperative development process of critical low cost electronic embedded architectures.

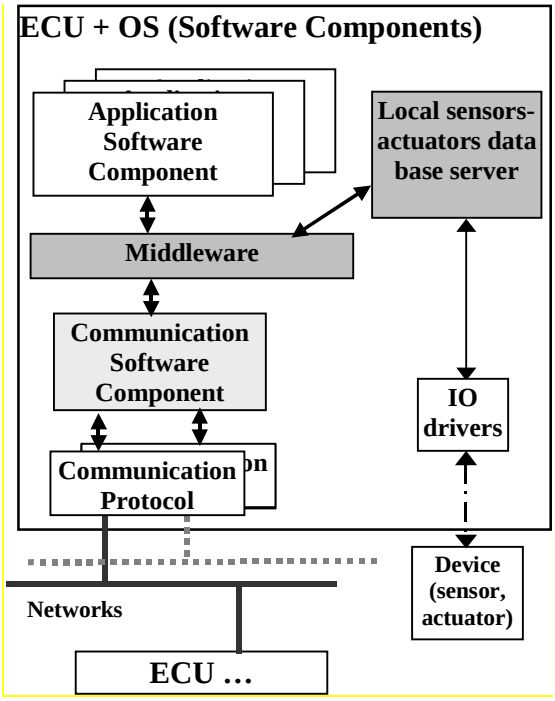

Fig. 1. Generic component – generic architecture

3. FROM DEDICATED COMPONENTS TO SYSTEM ARCHITECTURE

Nowadays, most embedded sub-systems (hardware and software) are separately defined and developed. Each one is dedicated to a single functionality and is designed and tested as closed systems by a supplier according to the carmaker requirements.

On the one hand, this is a bar to the reusability of solutions in other projects and on the other hand, this leads to oversize the resources (eg. number of ECUs, memory size, …). In order to solve this problem, several proposals have been made for the structuring of an embedded system into generic components and generic architecture and, also, for defining the "perimeter" of the what is re-usable and / or portable (a component, a set of component, …)

For example, the French AEE project (AEE, 1999) and the European ITEA project EAST-EEA (EAST, 2001) characterised formally the basic embedded components and defined the perimeter of the reusable ones. Furthermore they provided a generic architecture for an Electronic Control Unit (ECU), i.e.

a station connected to one or several network(s) and supporting the embedded application.

3.1. Component classes

The generic model presented in Fig. 1 enables the development of components that are independent of a specific ECU. That is the case for *Devices* (sensors and actuators) and *Application Software Components*. This independence is provided by two kinds of components:

- *middleware* and *communication software component* that hide the distribution and specifically the ECU supporting an information producer; in fact, the middleware actually implements a global real time database,
- *sensors – actuators data base server* whose function is to manage local data produced or consumed locally by devices directly connected to the ECU

These two components furnish a common interface to Application Software Components and on the other hand they implement fault tolerant services.

The communication between ECU and external actors are done thanks to *Communication Protocol Components* (data link layer for each connected network) and *I/O Drivers* that manage Input and Output for each device.

All the components, except devices, are implemented in software; they are supported by *ECU hardware* and managed by an Operating System. So the last class of component is *OS Software Components*.

3.2. Critical components

The safety properties of an embedded application depends on two points:

- reliability of the hardware architecture,
- properties on application behaviour.

For the first point, several metrics are available in order to evaluate this reliability. For example, starting with values given by furnishers of hardware components, we can compute the probability that a hardware architecture fails in one hour or the mean time between failures. Consequently, it allows to verify that these values are less than an imposed one; for example, a probability of failure occurrence in one hour that is less than 10^{-9} ensures that the given architecture is relevant to SIL (System integrity Level) class 4 (IEC, 1997).

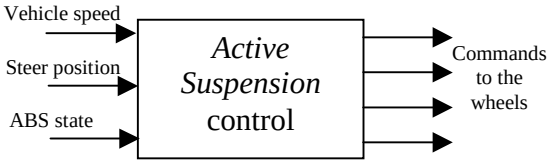

Fig. 2. Functional "Active Suspension Control" component

Note that this kind of evaluation is a necessary activity but not a sufficient one and the second point must be studied. It requires to have a quantified and precise model of the software behaviour: tasks, messages, scheduling policies, protocols, … For example, if we consider the power train or the chassis domain, the control laws impose stringent constraints that must be respected by an embedded architecture. For illustrate this, we analyse an “active suspension” that realises a balance between comfort and control. This function takes three input data: vehicle speed (VS), steer position (SP) and state of the ABS system (ABS) and delivers four consistent commands to the four wheels (FL, FR, RL, RR) as shown in Fig. 2. The temporal properties that an implementation of this function must respect, assumed here to be realized by only one task, are of the following types (Fig. 3):

- deadline on the response time of the algorithm implementing the control law,
- freshness of the input data when consumed by this algorithm,
- deadline on the response time of the message transmitting the output data to the wheel,
- temporal consistency of the three input data,
- temporal consistency of the four output data that have to be deliver in the same temporal window.

How to specify and how to ensure these properties ? We list below some open issues.

- *Time Triggered Approach*. Of course, the underlying products and techniques used in this approach allows a deterministic proof of the distributed application (TTP, 1999; Kopetz, 2002). Furthermore, by this way, fault detection and fault tolerant mechanisms are easier to introduce. Nevertheless, some problems must be addressed. In fact, if the specification of task activation is naturally deterministic and periodic for a control law, it is not the case for the sampling of driver solicitations (lights up, …). The overload due to systematic periodic task activation or message transmission might be compatible with the strong cost pressure imposed by the automotive industry.

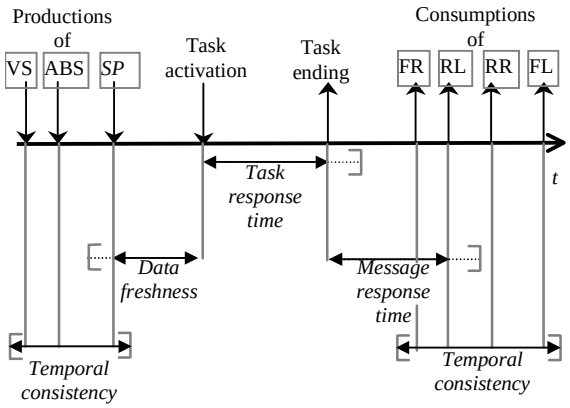

Fig. 3. Temporal constraints

There is a lack of flexibility and ability to extend and modify the embedded architectures. Finally, a significant part of an embedded application is not subject to strict hard real time constraints. So a compromise between ensuring "hard real time" constraints and "soft real time" constraints has to be found. A first response is given by protocols that allow a certain flexibility, as Flexray (Flexray, 2001), TTCAN (Führer *et al.*, 2000; Hartwich *et al.*, 2000), FTTCAN (Almeida and Pedreiras, 2000) or by Operating Systems as OSEK Time (OSEK-VDX, 2001) that keeps slots for non deterministic tasks. A second one relies on specific scheduling techniques as the *(m,k)* firm approach that guaranties always *m* satisfied deadline among *k* (Hamadoui and Ramanathan, 1995). Here the problem is the determination of *m* and *k*. Finally, deterministic approach and stochastic approach must be simultaneously used.

- *Robustness*. A vehicle, in particular the embedded electronic, is subject to non permanent failures due to electromagnetic perturbations; the safety properties must be kept in these situations. The problem is here to identify and model realistic perturbation scenarios (Navet *et al.*, 2000) and to propose a robust architecture (Gaujal and Navet, 2003) that is an architecture respecting the safety constraints even under these perturbations.

- *Portability vs Safety*. As we saw previously, the middleware is a central component for the portability of any embedded applications. Its main purpose is to act as a server of a global real time data base. Furthermore, for economical reasons, it has to offer a support for portable components, extensible applications, … Therefore, an in-car embedded middleware has to be sized and adapted to the application requirements. The solution of this problem is relevant to complex discrete optimisation techniques under real time constraints. These problems are known to be NP-complete and, consequently, efficient heuristics have to be developed (Santos-Marques *et al.*, 2003).

4. ARCHITECTURE REFERENCE MODELS AND DEVELOPMENT PROCESS

4.1. Development of a safe electronic embedded system

The quality of an embedded system depends obviously of the quality of its development. It is well known that only testing the final product for ensuring that it respects its functional requirement under the required properties is not the best way to develop a complex system because it implies costly back loops on former design steps. For an efficient process, validation and verification activities must take place at each step, from end user requirements to final implementation.

This leads to dispose of models of this system according to specific validation / verification techniques and consequently to the suited formalisms. Two remarks have to be done:

- *Several models*. Validation / verification techniques are not the same at each level of the process; they don't use the same modelling language or formalism and are not applied to same entities. For example, at end user requirement specification, a V&V activity can consist to prove the consistency of these requirements. The used formalism can be UML Use Cases, Message Sequence Charts, UML Collaboration Diagrams, … No information about software or hardware components or about distribution are used for this activity. On the other hand, to verify that a proposed implementation will respect performance and temporal properties needs to develop, just before coding, models that can represent event occurrences in a quantified way. For this purpose, well suited formalisms are Timed Automata, Temporal Petri Nets, Queuing Systems, … The modelled objects are tasks, frames, protocols, schedulers, memories, …
- *Modelling complexity*. Designers that have a good knowledge of electronic embedded systems and specific constraints in automotive domain are usually not specialist of the above mentioned languages or formalisms. This is a bar to the massive use of formal techniques.

A way to take account of these two points is to provide a domain oriented language that helps the designer to describe its system with its proper vocabulary and add a strict semantic to this language in order to furnish an automatic generation of formal models. At the present a lot of works are engaged in this way and main of them propose specific UML Profiles for achieving this purpose (OMG, 1999a; OMG, 1999b; Apvrille *et al.*, 2001; Cavaliere *et al.*, 2001). Fig. 4 shows schematically how these principles can be used.

For the description of component and architecture of component, computer science community engaged studies for the development of Architecture Description Languages (Taylor and Medvidovic, 1997). Unfortunately, these languages are restricted to software domains. Few of them are concerned with discrete event applications (Luckham, 1996; Allen and Garlan, 1997) or real time properties of an implementation (Vestal, 1993; Vestal,1995).

4.2. Reference Architecture Models

As explained in the previous section, a unique domain oriented language allows the representation of a system at each level of its development. This language is a declarative language for the modelling of components and architecture of components. At each design step, must be defined which component must be represented (classes and attributes) and how these components can be "composed" to form an architecture (relation, interaction, composition, …).

Furthermore, for traceability and model consistency purposes, the language has to provide a way for a formal description of the relation between components and architectures at different steps.

In order to illustrate these concepts we propose below an abstract of such a language, AIL_Transport (Elloy and Simonot-Lion, 2002) that was developed in the AEE Project (AEE, 1999). AIL_Transport has been defined as a modelling language dedicated to the specification of architectures described as an assembly of standard components. Every component is an instance of class belonging to a generic model, and it includes all pertinent characteristics necessary to the subsequent analysis of the whole architecture: interaction consistency, logical behavior, real-time performances, fault tolerant properties, … AIL_Transport is formalized using UML syntax.

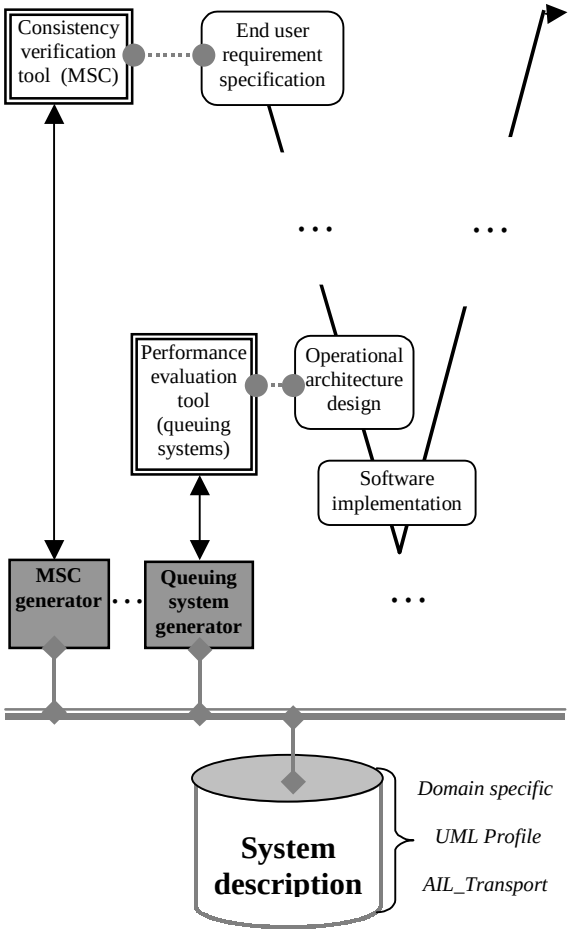

Fig. 4. A unique language - several validation activities

Using AIL_Transport, any designer describes the representation of an embedded architecture according to five different levels of abstraction. Each level models a particular point of view of the architecture. Entities specified at high levels ("vehicle project" level and "functional" level) are abstract components. Entities specified at low levels are, on the one hand, ECUs and communication protocols at the "hardware" level, on the other hand, the software and the devices (sensors and actuators) in the "software" level and the "operational" level. At each level, the designer describes the architecture as an assembly of objects instanced from predefined classes; then he specifies interactions between these objects using predefined connection types. The Fig. 5 illustrates these levels and their relationships. The development process associated to this representation links components introduced at different levels.

Vehicle Project Level. The upper level describes an embedded application from a vehicle point of view. At this level, objects shall represent functionalities offered by a vehicle (ABS, cruise control, air conditioning, etc), the different variant of these services (manual or automatic climate control system, for example) and the set of "on the shelf" vehicle versions. Five main classes are used at this architecture level: `Vehicle project`, `Vehicle type`, `Vehicle`, `Service`, `Variant`. Mainly, these classes document the architecture and support the project validation in terms of model consistency.

Functional Level. After building a validated vehicle model, the functional level describes one (or several) graph of elementary functional components realizing the services specified at the vehicle project level. Every graph of these components is specified disregarding the distribution and implementation aspects. The model supports a hierarchical specification of functions and flows. `Function`, `Functional Flow` and `Functional Architecture` are the main classes used to build graphs of elementary functions. Documentation, formal validation and formal test generation are the main process activities fulfilled at this step.

Hardware Level. This level models the electronic components of architecture as a set of processors, micro-controllers, electronic devices connected by networks. The main classes are

- *Operating Hardware*. Objects whose main subclasses are ECU (Electronic Control Unit for the computation nodes) and Network
- *Dependent Software Component*. These components are closely linked to hardware devices: Network Protocol, OS Software Components, I/O Drivers
- *Hardware Architecture* that specifies how each node is connected on one or several networks.

Software Level. At this level, two sets of classes are used. The first one is derived by class refinement of the functional architecture: the `Application Software Components`, the `Software Flow`, the `Instrumentation Hardware Objects` (Sensor and Actuators) and the `Software Architecture`. The second set of classes models the distribution of all software entities. For this, `Software Component` are decomposed in `Logical Task` communicating using `Software Input` and `Software Output` which are linked to `Software Flow` of the functional level. Moreover, the activation policies of `Logical Tasks` are specified (timed or event triggered).

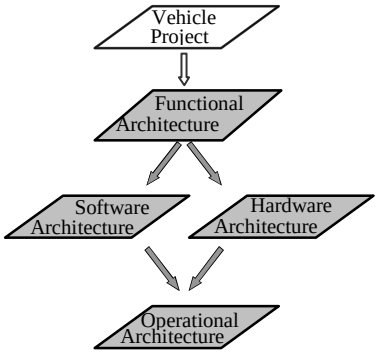

Fig. 5. Reference architectures during the design process

In order to validate the approach, a prototype implementing the access to AIL_Transport compliant data base has been realised. Several model generators were developed: synchronised timed automata based model for performance evaluation of an operational architecture and graph-based model for automatic allocation and scheduling of periodic distributed activities.

5. CONCLUSIONS

The introduction of software based systems are growing in automotive domain. This new technology has to be mastered in an efficient way. First it must preserve the know-how of each actor (car makers and suppliers), ensure the ability of system extensions or modifications and allow the portability of components. Furthermore, several constraints are applied to their design process: on the one hand, the development length must be shortened and, on the other hand, the safety of the system has to be formally proved as soon as possible before the real implementation. In this paper, we presented two axis for research and development activities in this context: the first one is concerned by on line architectures and mechanisms while the second one is relevant to off line a priori modelling, validation and verification.

The author would like to thank the actors of AEE Projecs, in particular, Jörn Migge, Franck Gasnier, Jean-Pierre Elloy, Yvon Trinquet, Xavier Hanin, Evelyne Silva, Bernard Bavoux, Philippe Germanicus for their essential contributions to the definition of the AIL_Transport language and Nicolas Navet for the fruitful debates that we had on this topic.

6. BIBLIOGRAPHY

TTP, (1999) http://www.tttech.com

Kopetz H., (2002), Time-Triggered Real-Time Computing, Proceedings of 15th IFAC World Congress, IFAC B'02, Barcelone, 21-26 juillet 2002.

OSEK-VDX, www.osek-vdx.org/

Navet N., Song Y-Q., Simonot F., (2000), Worst-Case Deadline Failure Probability in Real-Time Applications Distributed over CAN (Controller Area Network)", Journal of Systems Architecture, Elsevier Science, vol.46, n°7, 2000.

Gaujal B., Navet N., (2003), Optimal Replica Allocation for TTP/C Based Systems, to appear in Proc. 5th FeT IFAC Conference, FeT'2003, Fieldbus Technology, Aveiro (Portugal), 7-8 Juillet 2003.

Santos-Marquez R., Navet N., Simonot-Lion F., (2003), Frame Packing under Real-Time Constraints", to appear in Proc. 5th FeT IFAC Conference, FeT'2003, Fieldbus Technology, Aveiro (Portugal), 7-8 Juillet 2003

Flexray, (2001), http://www.flexray-group.com/

Führer T., B. Müller, W. Dieterle, F. Hartwich, R. Hugel, (2000) M. Walter, Time Trigger Communication on CAN (Time Trigger CAN-TTCAN) 7th International CAN Conference (ICC), Amsterdam, Netherlands, Oct. 24-25, 2000

Hartwich F., B. Müller, T. Führer, R. Hugel, (2000) CAN Network with Time Triggered Communication, http://www.can.bosch.com

Pedreiras P., L Almeida (2000), Combining Event-Triggered and Time-Triggered Traffic In FTT-CAN : Analysis of the Asynchronous Messaging System 2000 IEEE International Workshop on Factory Communication Systems (WFCS), Porto, Portugal, Sept. 6-8, 2000

Hamadoui M., Ramanathan P., (1995), A Dynamic Priority Assignment Technique for Streams with (m,k) firm deadlines, IEEE Transactions on Computers, 44(4), 1443-1451, December 1995.

IEC (1997), International Electrotechnical Commssion, 61508-1, Functional Safety of Electrical : Electronic / Programmable Electronic Safety-Related Systems.

AEE (1999), Architecture Electronique Embarquée, http://aee.inria.fr

EAST EEA (2001), European ITEA Project - Embedded Electronic Architectures, www.east-eea.net/docs

OMG, (1999a), White Paper on the Profile mechanism v.1.0», Analysis and Design Platform Task Force, Report ad/99-04-07, Avril 1999.

OMG, (1999b) UML Profile for Scheduling, Performance, and Time , RFP ad/99-03-13, 1999.

Apvrille L., P. de Saqui-Sannes, C. Lohr, P. Sénac, J.-P. Courtiat, "A New UML Profile for Real-time System Formal Design and Validation", Proceedings of the Fourth International Conference on the Unified Modeling Language (UML'2001), Toronto, Canada, October 2001.

Cavaliere D., Simonot-Lion F., Song Y.-Q., Hembert O., A Component Model Approach for Modelling and Validation of an Automated Manufacturing System, in Actes de 8th IEEE International Conference on Emerging Technologies and Factory Automation, Antibes - Juan les Pins, 15-18 octobre 2001.

Taylor R. N. and N. Medvidovic, (1997). A Framework for Classifying and Comparing Architecture Description Languages, *Technical Report*, Department of Information and Computer Science, University of California, Irvine.

Luckham D. C., (1996). Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Orderings of Events, *in Proceedings DIMACS Partial Order Methods Workshop IV*, Princeton University.

Allen R. and D. Garlan, (1997). A Formal Approach for Architectural Connection, *PhD thesis*, school of Computer Science, Carnegie-Mellon University, Pittsburgh.

Vestal S., (1993). Scheduling and Communicating in MetaH, *in Proceedings of Real -Time System Symposium*, Rleigh-Durham (NC), p.194-200.

Vestal S., (1995). MetaH Reference Manual, *Technical Report*, Honeywell Technology Center.

Elloy J.-P., Simonot-Lion F., *An Architecture Description Language for In-Vehicle Embedded System development*, Proceedings of 15th IFAC World Congress, IFAC B'02, Barcelone, 21-26 juillet 2002.